УДК 004.056.53

**Коломыцев М.В., Носок С.А. (НТУУ «КПИ»)**

# АУТЕНТИФИКАЦИЯ В WEB-ПРИЛОЖЕНИЯХ

Современная тенденция создания WEB-приложений состоит в персонализации контента и управляемом доступе к предоставляемым сервисам. В связи с этим, повышаются требования к аутентификации клиентов WEB-сайтов. Существующие подходы к построению механизма аутентификации зачастую уязвимы к атакам. В статье рассматриваются различные ограничения и меры, обеспечение которых позволит снизить риск нарушений безопасности.

**Актуальность задачи безопасности систем аутентификации.** Консорциум Web Application Security Consortium периодически представляет статистику уязвимостей и угроз безопасности Web-приложений. В предлагаемой ими классификации угроз отдельный класс угроз связан с атаками на используемые методы аутентификации [1]. Связаны они с особенностями используемых в WEB схем аутентификации и их потенциальные ограничениями.

Ограничивающими факторами при выборе схемы аутентификации являются:

– ограничения на сложность реализации. Используемые технологии должны поддерживаться на разных платформах и не создавать существенной нагрузки на сервер. Поэтому технологии, реализующие сложные вычисления на стороне клиента (с использованием например, Javascript, Java, ActiveX и FlashB) используются не часто. Наиболее распространенной формой обмена аутентифицирующией информацией в последовательности HTTP запросов являются *cookie*.

– восприятие пользователями. Форма диалога должна быть максимально простой, и исключать необходимость установки дополнительных компонент на компьютере клиента, либо организации процедуры аутентификации в виде длинной последовательности диалоговых окон.

– производительность. Защищенные протоколы аутентификации, криптографические преобразования вообще, существенно снижают производительность сервера. Это в частности, касается протокола SSL.

**Методы аутентификации в WEB**

Базовая аутентификация. Это простой протокол проверки подлинности, поддерживаемый всеми браузерами [3]. Его еще иногда называют HTTP аутентификацией. Протокол работает следующим образом:

– ресурсы, для доступа к которым необходимо пройти аутентификации, помещаются в отдельный каталог, и создается файл конфигурации, описывающий местоположение базы учетных записей и набор ограничений.

– браузер отправляет HTTP запрос на доступ к защищенному ресурсу;

– серверный HTTP процесс определяет, что необходим доступ к защищенному ресурсу и считывает из файла конфигурации параметры. Выяснив, что пользовательский запрос не содержит аутентифицируещей информации, он его отвергает, сообщая клиенту, что требуется аутентификация с помощью пароля.

– браузер, получив отказ, проверят, нет ли в кеше пароля и логина для указанной области данного сервера. Если не находит, то выводит диалоговое окно для ввода имени и пароля. Введенные пользователем данные сохраняются для дальнейшего использования.

– браузер повторяет запрос к серверу, но уже с информацией о имени и пароле пользователя. Пароль пользователя передается в открытом виде.

– серверный HTTP процесс сверяет полученную информацию с той, которая хранится в базе учетных записей. Если учетная запись с таким именем не существует, или переданный пароль не совпадает с тем, который хранится в базе, сервер отвергает запрос. Получив отказ, браузер повторно выводить окно для ввода имени или пароля.

– если все проверки оказались успешными, сервер пересылает браузеру нужную HTTP страничку.

Теоретически, для доступа к каждой защищенной страничке необходимо указать имя и пароль. На практике, браузер пытается получить доступ к защищаемым страничкам, используя имя и пароль, хранимые в кеше браузера.

Недостатком протокола является пересылка пароля в открытом виде. В протоколе HTTP 1.1 пароль шифруется, но очень слабым алгоритмом (Base64). Другим, важным точки зрения безопасности, обстоятельством является то, что установленное доверительное соединение может быть разорвано только путем закрытия браузера. Даже если пользователь переключил браузер на сайт, не требующий аутентификации, имя и пароль хранятся в памяти. Таким образом, оставленный без присмотра компьютер может служит средством, для беспрепятственного доступа других пользователей к защищенным ресурсам.

**Аутентификация на основе хеша**

Данный метод аутентификации был добавлен в протокол HTTP, для исправления основных недостатков базовой аутентификации. Во многом этот метод работает так же, как и базовая аутентификация [4]. При попытке доступа браузера к защищенному ресурсу, запрос отвергается. Но в сообщении сервера указывается, что требуется аутентификация на основе хеша и содержится значение (*nonce*), индивидуальное для каждого запроса. Это значение может формироваться, например, по значению текущего времени и ip-адреса клиента. При вычислении хеша используется алгоритм MD5. Затем:

– браузер получает пароль пользователя (из диалогового окна или из памяти);

– объединяет в одну строку имя пользователя, имя области аутентификации и пароль, затем вычисляет хеш полученного значения (хешA);

– объединяет в одну строку запрашиваемый URL с названием метода, используемого в запросе (PUT, GET, …) и вычисляет хеш полученной строки (хешB);

– выполняет конкатенацию хешA и хешB со значением nonce и вычисляет результирующий хеш;

– полученное значение хеша отправляется серверу;

– когда сервер получает клиентский запрос, он извлекает из хранилища пароль пользователя, на его основе повторяет вычисления контрольных значений хеша. Возможна реализация алгоритма, когда хранится не пароль пользователя, а вычисленное значение хешA. Процедура аутентификации считается успешной, если результат вычислений совпадают с тем, что получено от клиента.

Протокол аутентификации на основе хеша нельзя считать полностью защищенным. Если трафик не шифруется с помощью SSL или HTTPS, то атакующий может просматривать все запросы в текстовом виде. Он может сохранить текст запроса целиком и позднее повторить его. Если запрашиваемым ресурсом является статическая HTML страница, то он получит ее копию (которую, впрочем, он и так мог получить, просто перехватывая трафик между клиентом и сервером). Если же HTML страницы формируются динамически, то ресурсом является сценарий сервера, и последствия такой атаки могут быть гораздо серьезнее. Решением этой проблемы может быть хранение отправленного клиенту значения *nonce* и запрет его повторного использования. Однако такой вариант является достаточно затратным для ресурсов сервера. Более простой ( и менее надежный) способ – формирование *nonce* на основе текущей даты и временной метки. HTTP сервер примет запрос на аутентификацию, только если он поступил из того же ip-адреса и не является слишком старым.

Другая уязвимость протокола состоит в том, что сервер ограничен в возможности защищенного хранения паролей. Поскольку для вычисления хеша используется пароль, сервер должен иметь доступ к паролю в открытом виде, либо он должен хранить вычисленное значение *хешA*. Таким образом, данный протокол нельзя использовать, если

пароли защищены с помощью необратимого шифрования. В этом смысле данный протокол более уязвим, чем протокол базовой аутентификации. Если неавторизованный пользователь получает доступ к базе паролей, он получает доступ к хешу, сформированному на основе имени пользователя и пароля. В этом случае, ему нет необходимости знать пароль, он может воспользоваться значением *хешА* для атаки данного сайта. Таким образом, безопасность базы паролей при использовании данного протокола является важнейшей задачей.

**Аутентификация на основе форм**

В этом протоколе для ввода имени и пароля пользователя используются HTML формы. Форма – это основной способ ввода информации на стороне клиента, для отправки ее серверу (точнее, Web-приложению). Стандартной реализации данного метода аутентификации нет. Разработчики сами определяют детали его функционирования. Введенные пользователем имя и пароль передаются на сервер, обычно по протоколу SSL или TLS.

Проблема безопасности HTML-форм заключается в том, что по своей природе Web-приложения приложениями «без предыстории» (stateless). Если для доступа к ресурсу пользователь должен быть аутентифицирован, то возникает необходимость сохранения учетных данных, введенных в поля формы, что бы передать на сервер как часть следующей транзакции. Что бы каким-то образом сохранить аутентифицирующую информацию, разработчики могут использовать два подхода – сохранять необходимую информацию на сервере либо на стороне клиента, причем большинство разработчиков предпочитают второй вариант, считая его более легким в реализации. Информация на стороне клиента может сохраняться в файлах *cookies* браузера, в виде значений, добавляемых к URL и в скрытых полях HTML-форм. Уязвимость последних двух вариантов очевидна [2].

В случае, если информация о текущей сессии пользователя хранится на сервере (в виде временных файлов или временных таблиц в базе данных), ссылки на эту информацию должны быть доступны на стороне клиента. Как правило, они хранятся в виде ***Id*** сессии в файлах *cookie*. *Cookie* могут использоваться сервером для опознания ранее аутентифицированных пользователей. В этом случае, сервер, по окончании процедуры аутентификации, отправляет страницу успешного входа, прикрепив *cookie* с неким идентификатором сессии. Эта *cookie* может быть действительна только для текущей сессии браузера, но может быть изменена и на длительное хранение. Каждый раз, когда пользователь запрашивает страницу с сервера, браузер автоматически отправляет *cookie* с идентификатором сессии серверу. Сервер проверяет идентификатор по своей базе идентификаторов и, при наличии в базе такого идентификатора, разрешает пользователю доступ.

Файлы *cookie* – это текстовые файлы, содержащие пары имя/значение. Кроме того, они могут содержать информацию о сроке действия, пути и доменном имени (RFC2965). Браузер MS Internet Explorer хранит их в виде отдельных файлов в папке *Application Data\cookies.* Браузеры на основе Netscape и Mozilla хранят их в одном файле, называемом *cookies.txt*. А поскольку они сохраняются в файл, то могут быть изменены с помощью текстового редактора.

Сессионные *cookie* подделать сложнее, однако существуют программные средства, например, *Burp Proxy*, которые позволяют просматривать и изменять сессионные *cookie*. Обзор атак на схемы аутентификации Web-приложений приведены в [2].

Рекомендации по построению схем аутентификации WEB –клиентов:

– Выбирать подходящий уровень защищенности. Чем выше уровень защищенности программной системы, тем сложнее она в реализации и управлении. Нет необходимости использовать сложные схемы аутентификации, например, для новостных сайтов. Выбор уровня защищенности можно рассматривать как компромисс между удобством пользователя, производительностью и защищенностью;

– Не изобретать велосипед. Практика показывает, что системы, созданные людьми, не обладающими достаточной практикой, подвержены многочисленным уязвимостям. Нужно использовать проверенные решения;

– Не полагаться на секретность протокола аутентификации. Акцент необходимо делать на секретности криптографических ключей. Конечно, афишировать подробности алгоритмы протокола так же не стоит;

– Учитывать все особенности используемых криптографических методов и протоколов. При реализации криптографических алгоритмов, таких как протокол формирования хеша SHA-1, протокол аутентификации сообщений HMAC, протоколы высокого уровня, например, SSL необходимо понимать круг задач, решаемых с их помощью. Так, с помощью SSL можно аутентифицировать пользователей, однако для этого необходимо, что бы у клиента был сертификат по стандарту X.509, следовательно, необходимо разворачивать инфраструктуру PKI. Для большинства Web-сайтов такой вариант не подходит;

– Ограничить видимость паролей. Хранение паролей в скрытых полях форм, их передача в открытом виде не должны использоваться. Пароль, введенный пользователем, должен передаваться с использование протокола SSL, и ни в коем случае не передаться по протоколу HTTP. При успешном завершении процедуры аутентификации сервер должен отослать на клиентский компьютер cookie с идентификатором сессии, который сложно предугадать;

– Запретить использование легко подбираемых паролей. Существует множество рекомендаций, о том, каким пароль не должен быть, что бы избежать атак подбора по словарю или угадывания пароля на основе личной информации пользователя;

– Обязательно повторять процедуру аутентификации перед сменой пароля. В противном случае, атакующий может инициировать смену пароля, просто воспроизведя информацию из cookie;

– Безопасное манипулирование аутентификаторами. Аутентификаторы (cookie) используются клиентским приложением для доступа к серверу. Знание аутентификатора позволяет атакующему обойти систему авторизации сервера и получить доступ к его ресурсам. Поэтому аутентификаторы должны генерироваться таким образом, что бы их значение нельзя было предсказать. Поскольку cookie может содержать и другую конфиденциальную информации, он должен быть защищен от подмены. Сделать это можно, используя цифровую подпись. Для того, что бы пользователь не мог подделать цифровую подпись, необходимо добавить к ЭЦП некий секретный компонент, который пользователь не может видеть. Это делается с помощью секретного ключа, хранимого на сервере. Такая конструкция называется "message authentication code" (MAC), а стандартный способ ее применения называется HMAC.

– Не использовать постоянные (persistent) cookie. В отличие от сессионных (temporary) cookie, которые хранятся только в оперативной памяти, постоянные хранятся на диске. Нарушение безопасности клиентского компьютера может привести к раскрытию их содержимого.

– Ограничить время жизни аутентификатора. Поскольку cookie хранятся на компьютере клиента, они могут быть изменены злоумышленником, срок их жизни может быть продлен. Например, в браузере Netscape это можно сделать простым редактированием текстового файла. Что бы избежать этого, необходимо добавлять в cookie криптографически защищенную временную метку, либо хранить срок его действия на стороне сервера.

## Литература

1. The WASC Threat Classification v2.0. http://projects.webappsec.org/w/page/13246978/ Threat-Classification

2. Authentication and Session Management on the Web http://www.westpoint.ltd.uk/ advisories/ Paul_Johnston_GSEC.pdf
3. Web Authentication Security http://www.sans.org/reading_room/whitepapers/webservers/ web-authentication-security_1250
4. A Guide to Web Authentication Alternatives http://unixpapa.com/auth/

Надійшла: 02.06.2011 р.

Рецензент: д.т.н., проф. Петров О.С.